Estruturas de Dados - ESP412

Profª Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal de Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

November 28, 2024

Algoritmos de Ordenação

Algoritmos de Ordenação

SelectionSort

Ordenação por Seleção (*SelectionSort*)

A Ordenação por Seleção (*SelectionSort*) é um algoritmo simples, que realiza uma ordenação através da seleção do menor elemento de uma sequência e da colocação deste elemento na primeira posição, repetindo este processo com a sequência não ordenada.

Ordenação por Seleção (*SelectionSort*)

A Ordenação por Seleção (*SelectionSort*) é um algoritmo simples, que realiza uma ordenação através da seleção do menor elemento de uma sequência e da colocação deste elemento na primeira posição, repetindo este processo com a sequência não ordenada.

Se você pedir para alguém que ordene uma sequência de números qualquer, há uma probabilidade alta dessa pessoa aplicar o *SelectionSort*.

Ordenação por Seleção (*SelectionSort*)

A Ordenação por Seleção (*SelectionSort*) é um algoritmo simples, que realiza uma ordenação através da seleção do menor elemento de uma sequência e da colocação deste elemento na primeira posição, repetindo este processo com a sequência não ordenada.

Se você pedir para alguém que ordene uma sequência de números qualquer, há uma probabilidade alta dessa pessoa aplicar o *SelectionSort*.

Assim como o *InsertionSort* e o *BubbleSort*, o *SelectionSort* tem custo $O(N^2)$ no pior caso e $O(N)$ no melhor caso [IME 2023].

Ordenação por Seleção (*SelectionSort*)

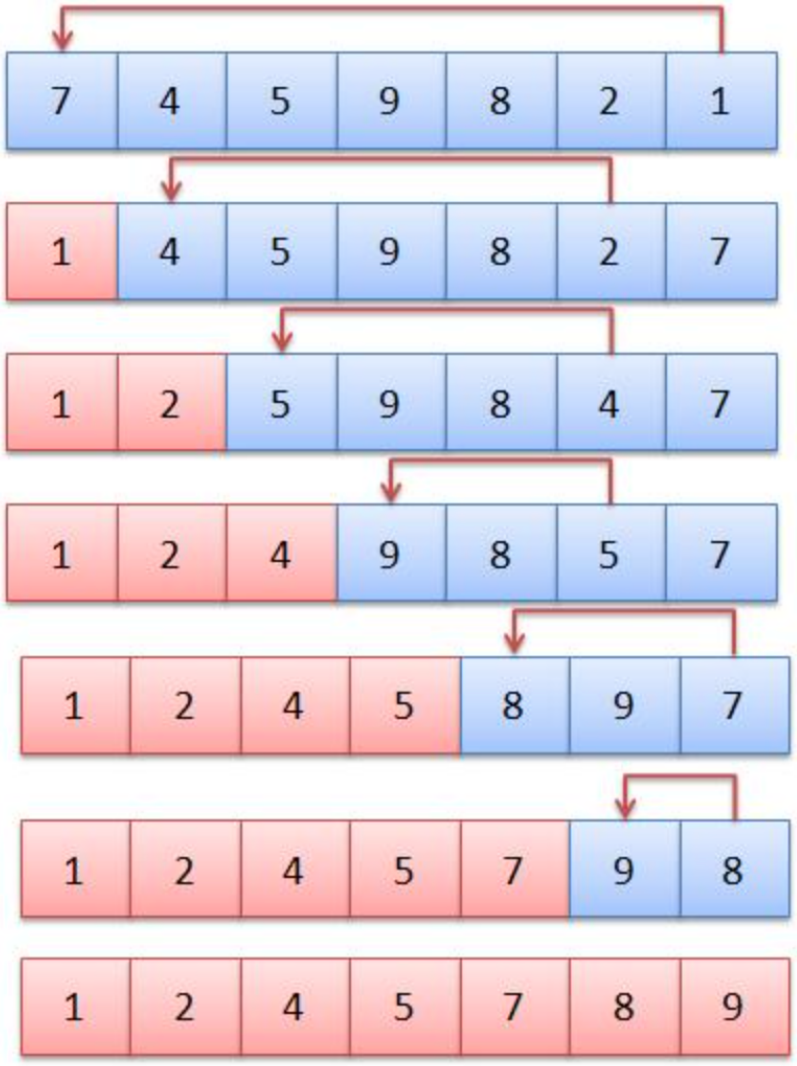
A Ordenação por Seleção (*SelectionSort*) é um algoritmo simples, que realiza uma ordenação através da seleção do menor elemento de uma sequência e da colocação deste elemento na primeira posição, repetindo este processo com a sequência não ordenada.

Se você pedir para alguém que ordene uma sequência de números qualquer, há uma probabilidade alta dessa pessoa aplicar o *SelectionSort*.

Assim como o *InsertionSort* e o *BubbleSort*, o *SelectionSort* tem custo $O(N^2)$ no pior caso e $O(N)$ no melhor caso [IME 2023].

Vejamos um vídeo explicativo do algoritmo *SelectionSort* no Youtube: `https://www.youtube.com/watch?v=lyZQPjUT5B4`

Selection Sort:



Ordenação por Seleção → *SelectionSort*

Algorithm void selectionSort(*int tam*, *int v*[])

```
1:  for i = 0 to N do
2:      menor = i;
3:      for j = i to N do
4:          if v[j] < v[menor] then
5:              menor = j;
6:          end if
7:      end for
8:      aux = v[menor];
9:      v[menor] = v[i];
10:     v[i] = aux;
11: end for
```

Ordenação por Seleção → *SelectionSort*

```
void selectionSort(int tam, int v[]){
    int aux, menor;
    for (int i = 0; i < tam; i++){
        menor = i;
        for(int j = i; j < tam; j++)
            if (v[j] < v[menor])
                menor = j;

        aux = v[menor];
        v[menor] = v[i];
        v[i] = aux;
    }
}
```

MergeSort

Ordenação por Junção/Mistura (*MergeSort*)

MergeSort é um algoritmo eficiente de ordenação por divisão e conquista. Se nossa missão é ordenar um vetor comparando seus elementos, então nenhum algoritmo de ordenação é mais veloz do que $nlog_n$.

Ordenação por Junção/Mistura (*MergeSort*)

MergeSort é um algoritmo eficiente de ordenação por divisão e conquista. Se nossa missão é ordenar um vetor comparando seus elementos, então nenhum algoritmo de ordenação é mais veloz do que $nlog_n$.

O funcionamento do *MergeSort* baseia-se em uma rotina fundamental cujo nome é **Merge**. Sucessivas execuções de merge ordena o vetor.

Ordenação por Junção/Mistura (*MergeSort*)

MergeSort é um algoritmo eficiente de ordenação por divisão e conquista. Se nossa missão é ordenar um vetor comparando seus elementos, então nenhum algoritmo de ordenação é mais veloz do que $nlog_n$.

O funcionamento do *MergeSort* baseia-se em uma rotina fundamental cujo nome é **Merge**. Sucessivas execuções de merge ordena o vetor.

Sua eficiência é $O(nlog_n)$ para o melhor, pior e para o caso médio [D.E. Knuth 1973].

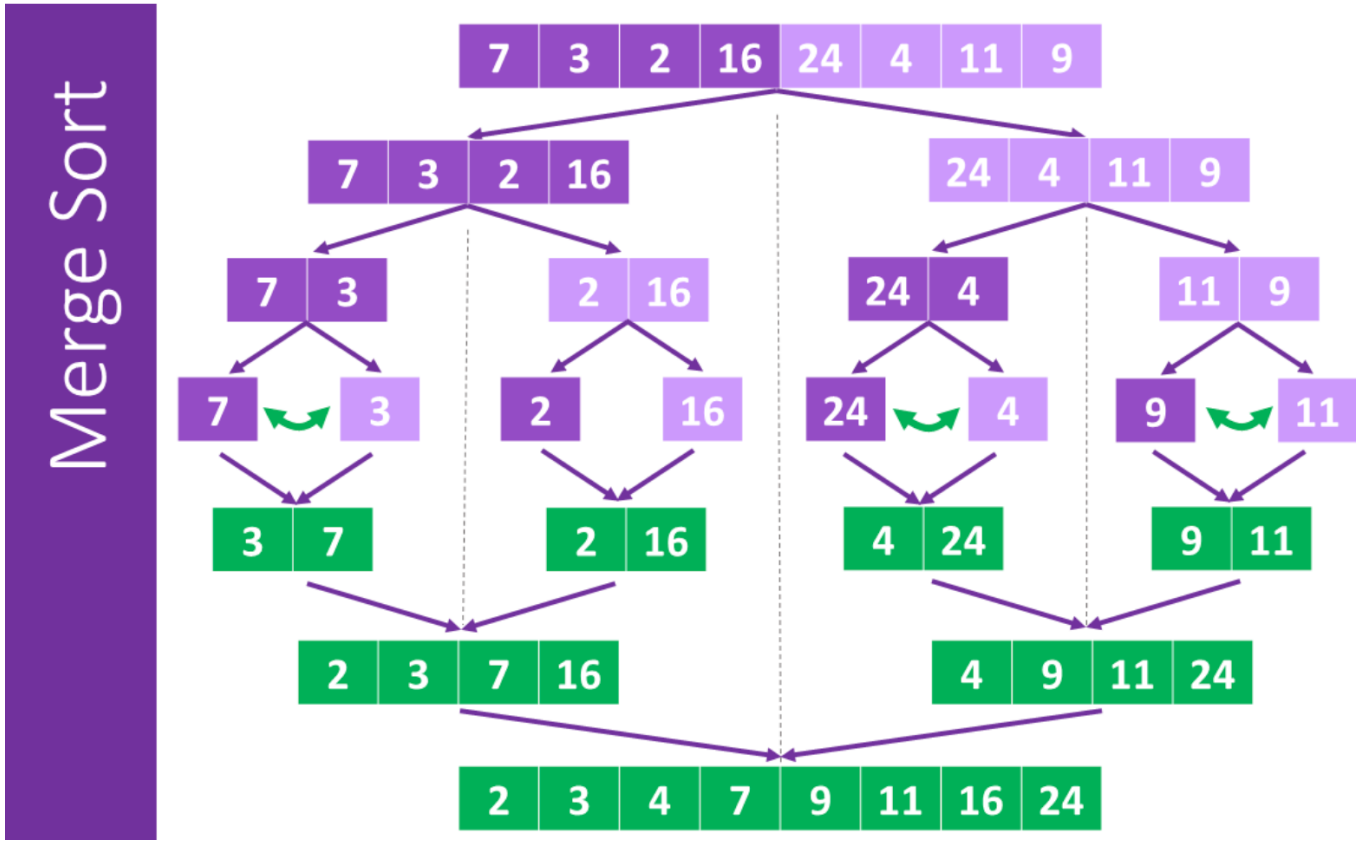
Ordenação por Junção/Mistura (*MergeSort*)

MergeSort é um algoritmo eficiente de ordenação por divisão e conquista. Se nossa missão é ordenar um vetor comparando seus elementos, então nenhum algoritmo de ordenação é mais veloz do que $nlog_n$.

O funcionamento do *MergeSort* baseia-se em uma rotina fundamental cujo nome é **Merge**. Sucessivas execuções de merge ordena o vetor.

Sua eficiência é $O(nlog_n)$ para o melhor, pior e para o caso médio [D.E. Knuth 1973].

Vejamos um vídeo explicativo do algoritmo *MergeSort* no Youtube: `https://www.youtube.com/watch?v=XaqR3G_NVoo`



Vejamos uma simulação do algoritmo *MergeSort*:
`https://www.101computing.net/Merge-Sort-algorithm/`

Ordenação por Junção/Mistura → *MergeSort*

Algorithm void mergeSort($int\ V[], int\ l, int\ r$)

1: **if** $(l < r)$ **then**
2: $m = (l + r)/2;$
3: $mergeSort(V, l, m)$
4: $mergeSort(V, m + 1, r)$
5: $merge(V, l, m, r)$
6: **end if**

Ordenação por Junção/Mistura → *MergeSort*

Algorithm void merge($int\ V[], int\ l, int\ m, int\ r$)

```
1:  n1 = m − l + 1;
2:  n2 = r − m;
3:  for i = 0 to n1 do
4:      L[i] = V[l + i];
5:  end for
6:  for j = 0 to n2 do
7:      R[j] = V[m + 1 + j];
8:  end for
9:  i = 0; j = 0;
10: k = l;
11: CONTINUA ···
```

Ordenação por Junção/Mistura → *MergeSort*

Algorithm void merge($int\ V[], int\ l, int\ m, int\ r$)

```
1:  ··· CONTINUAÇÃO
2:  while (i < n1)and(j < n2) do
3:      if (L[i] <= R[j]) then
4:          V[k] = L[i];
5:          i++;
6:      else
7:          V[k] = R[j];
8:          j++;
9:      end if
10:     k++;
11: end while
12: CONTINUA ···
```

Ordenação por Junção/Mistura → *MergeSort*

Algorithm void merge($int\ V[], int\ l, int\ m, int\ r$)

```
1: ··· CONTINUAÇÃO
2: while (i < n1) do
3:     V[k] = L[i];
4:     i++;
5:     k++;
6: end while
7: while (j < n2) do
8:     V[k] = R[j];
9:     j++;
10:    k++;
11: end while
```

Referências

Referências


D.E. Knuth. *The Art of Computer Programming*. [S.l.]: Addison-Wesley, 1973. v. 1 - 3.


IME. *Aulas de Estruturas de Dados*. 2023. `https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/index.html`. [Online; accessed 13-February-2023].